

python / cpython Public

<> Code Issues 5k+ Pull requests 2.2k Actions Projects Security and q

Commit 8392b2f



miss-islington and serhiy-storchaka authored on Oct 8, 2025 · ✖ 57 / 69 · Verified



[3.12] [gh-139700](#): Check consistency of the zip64 end of central directory record ([GH-139702](#)) ([GH-139708](#)) ([GH-139712](#))

(cherry picked from commit [333d4a6](#))

Support records with "zip64 extensible data" if there are no bytes prepended to the ZIP file.

(cherry picked from commit [162997b](#))

Co-authored-by: Serhiy Storchaka <storchaka@gmail.com>

[3.12](#) (#139712) · v3.12.13 v3.12.12

1 parent [dea7e3d](#) commit 8392b2f

3 files changed +113 -23 lines changed

Top

- ✓ Lib
 - ✓ test/test_zipfile
 - test_core.py
 - ✓ zipfile
 - __init__.py
- ✓ Misc/NEWS.d/next/Security
 - 2025-10-07-19-31-34.gh-issue-139700.vNHU1O.rst

3 files changed +113 -23 lines changed

```

  ✓ Lib/test/test_zipfile/test_core.py
  ↑↑ -885,6 +885,8 @@ def make_zip64_file(

```

```

885 885         self, file_size_64_set=False, file_size_extra=False,
886 886         compress_size_64_set=False, compress_size_extra=False,
887 887         header_offset_64_set=False, header_offset_extra=False,
888 888         +     extensible_data=b'',
889 889         +     end_of_central_dir_size=None, offset_to_end_of_central_dir=None,
888 890     ):
889 891         """Generate bytes sequence for a zip with (incomplete) zip64 data.
890 892
@@ -938,6 +940,12 @@ def make_zip64_file(
938 940
939 941         central_dir_size = struct.pack('<Q', 58 + 8 *
len(central_zip64_fields))
940 942         offset_to_central_dir = struct.pack('<Q', 50 + 8 *
len(local_zip64_fields))
943 943         +     if end_of_central_dir_size is None:
944 944         +         end_of_central_dir_size = 44 + len(extensible_data)
945 945         +     if offset_to_end_of_central_dir is None:
946 946         +         offset_to_end_of_central_dir = (108
947 947         +             + 8 * len(local_zip64_fields)
948 948         +             + 8 * len(central_zip64_fields))
941 949
942 950         local_extra_length = struct.pack("<H", 4 + 8 *
len(local_zip64_fields))
943 951         central_extra_length = struct.pack("<H", 4 + 8 *
len(central_zip64_fields))
@@ -966,14 +974,17 @@ def make_zip64_file(
966 974         + filename
967 975         + central_extra
968 976         # Zip64 end of central directory
969 969         -     + b"PK\x06\x06,\x00\x00\x00\x00\x00\x00\x00-\x00-"
970 970         -     + b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00"
977 977         +     + b"PK\x06\x06"
978 978         +     + struct.pack('<Q', end_of_central_dir_size)
979 979         +     + b"- \x00-
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00"
971 980         + b"\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00"
972 981         + central_dir_size
973 982         + offset_to_central_dir

```

	983	+	+ extensible_data
974	984		# Zip64 end of central directory locator
975		-	+ b"PK\x06\x07\x00\x00\x00\x001\x00\x00\x00\x00\x00\x01"
976		-	+ b"\x00\x00\x00"
	985	+	+ b"PK\x06\x07\x00\x00\x00\x00"
	986	+	+ struct.pack('<Q', offset_to_end_of_central_dir)
	987	+	+ b"\x01\x00\x00\x00"
977	988		# end of central directory
978	989		+ b"PK\x05\x06\x00\x00\x00\x00\x01\x00\x01\x00:\x00\x00\x002\x00"
979	990		+ b"\x00\x00\x00\x00"
			@@ -1004,6 +1015,7 @@ def test_bad_zip64_extra(self):
1004	1015		with self.assertRaises(zipfile.BadZipFile) as e:
1005	1016		zipfile.ZipFile(io.BytesIO(missing_file_size_extra))
1006	1017		self.assertIn('file size', str(e.exception).lower())
	1018	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_file_size_extra)))
1007	1019		
1008	1020		# zip64 file size present, zip64 compress size present, one field in
1009	1021		# extra, expecting two, equals missing compress size.
			@@ -1015,6 +1027,7 @@ def test_bad_zip64_extra(self):
1015	1027		with self.assertRaises(zipfile.BadZipFile) as e:
1016	1028		zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1017	1029		self.assertIn('compress size', str(e.exception).lower())
	1030	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1018	1031		
1019	1032		# zip64 compress size present, no fields in extra, expecting one,
1020	1033		# equals missing compress size.
			@@ -1024,6 +1037,7 @@ def test_bad_zip64_extra(self):
1024	1037		with self.assertRaises(zipfile.BadZipFile) as e:
1025	1038		zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1026	1039		self.assertIn('compress size', str(e.exception).lower())
	1040	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1027	1041		
1028	1042		# zip64 file size present, zip64 compress size present, zip64 header
1029	1043		# offset present, two fields in extra, expecting three, equals
			missing
			@@ -1038,6 +1052,7 @@ def test_bad_zip64_extra(self):

```

1038 1052         with self.assertRaises(zipfile.BadZipFile) as e:
1039 1053             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1040 1054         self.assertIn('header offset', str(e.exception).lower())
1055 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1041 1056
1042 1057         # zip64 compress size present, zip64 header offset present, one field
1043 1058         # in extra, expecting two, equals missing header offset
@@ -1050,6 +1065,7 @@ def test_bad_zip64_extra(self):
1050 1065         with self.assertRaises(zipfile.BadZipFile) as e:
1051 1066             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1052 1067         self.assertIn('header offset', str(e.exception).lower())
1068 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1053 1069
1054 1070         # zip64 file size present, zip64 header offset present, one field in
1055 1071         # extra, expecting two, equals missing header offset
@@ -1062,6 +1078,7 @@ def test_bad_zip64_extra(self):
1062 1078         with self.assertRaises(zipfile.BadZipFile) as e:
1063 1079             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1064 1080         self.assertIn('header offset', str(e.exception).lower())
1081 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1065 1082
1066 1083         # zip64 header offset present, no fields in extra, expecting one,
1067 1084         # equals missing header offset
@@ -1073,6 +1090,63 @@ def test_bad_zip64_extra(self):
1073 1090         with self.assertRaises(zipfile.BadZipFile) as e:
1074 1091             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1075 1092         self.assertIn('header offset', str(e.exception).lower())
1093 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1094 +
1095 +         def test_bad_zip64_end_of_central_dir(self):
1096 +             zipdata = self.make_zip64_file(end_of_central_dir_size=0)
1097 +             with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1098 +                 zipfile.ZipFile(io.BytesIO(zipdata))
1099 +             self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1100 +
1101 +             zipdata = self.make_zip64_file(end_of_central_dir_size=100)

```

```
1102 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1103 +             zipfile.ZipFile(io.BytesIO(zipdata))
1104 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1105 +
1106 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=0)
1107 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1108 +             zipfile.ZipFile(io.BytesIO(zipdata))
1109 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1110 +
1111 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=1000)
1112 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*locator'):
1113 +             zipfile.ZipFile(io.BytesIO(zipdata))
1114 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1115 +
1116 +     def test_zip64_end_of_central_dir_record_not_found(self):
1117 +         zipdata = self.make_zip64_file()
1118 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1119 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1120 +             zipfile.ZipFile(io.BytesIO(zipdata))
1121 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1122 +
1123 +         zipdata = self.make_zip64_file(
1124 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1125 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1126 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1127 +             zipfile.ZipFile(io.BytesIO(zipdata))
1128 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1129 +
1130 +     def test_zip64_extensible_data(self):
1131 +         # These values are what is set in the make_zip64_file method.
1132 +         expected_file_size = 8
1133 +         expected_compress_size = 8
1134 +         expected_header_offset = 0
1135 +         expected_content = b"test1234"
1136 +
1137 +         zipdata = self.make_zip64_file(
1138 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1139 +         with zipfile.ZipFile(io.BytesIO(zipdata)) as zf:
1140 +             zinfo = zf.infolist()[0]
1141 +             self.assertEqual(zinfo.file_size, expected_file_size)
```

```

1142 +         self.assertEqual(zinfo.compress_size, expected_compress_size)
1143 +         self.assertEqual(zinfo.header_offset, expected_header_offset)
1144 +         self.assertEqual(zf.read(zinfo), expected_content)
1145 +         self.assertTrue(zipfile.is_zipfile(io.BytesIO(zipdata)))
1146 +
1147 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1148 +             zipfile.ZipFile(io.BytesIO(b'prepended' + zipdata))
1149 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(b'prepended' +
zipdata)))

```

```
1076 1150
```

```
1077 1151     def test_generated_valid_zip64_extra(self):
```

```
1078 1152         # These values are what is set in the make_zip64_file method.
```



Lib/zipfile/__init__.py



```
@@ -231,41 +231,57 @@ def is_zipfile(filename):
```

```
231 231         else:
```

```
232 232             with open(filename, "rb") as fp:
```

```
233 233                 result = _check_zipfile(fp)
```

```
234 -         except OSError:
```

```
234 +         except (OSError, BadZipFile):
```

```
235 235             pass
```

```
236 236         return result
```

```
237 237
```

```
238 238     def _EndRecData64(fpin, offset, endrec):
```

```
239 239         """
```

```
240 240         Read the ZIP64 end-of-archive records and use that to update endrec
```

```
241 241         """
```

```
242 -         try:
```

```
243 -             fpin.seek(offset - sizeEndCentDir64Locator, 2)
```

```
244 -         except OSError:
```

```
245 -             # If the seek fails, the file is not large enough to contain a ZIP64
```

```
242 +         offset -= sizeEndCentDir64Locator
```

```
243 +         if offset < 0:
```

```
244 +             # The file is not large enough to contain a ZIP64
```

```
246 245             # end-of-archive record, so just return the end record we were given.
```

```
247 246         return endrec
```

```
248 -
```

```
247 +         fpin.seek(offset)
```

```
249 248         data = fpin.read(sizeEndCentDir64Locator)
```

```

250 249         if len(data) != sizeEndCentDir64Locator:
251 -         return endrec
250 +         raise OSError("Unknown I/O error")
252 251         sig, diskno, reloff, disks = struct.unpack(structEndArchive64Locator,
data)
253 252         if sig != stringEndArchive64Locator:
254 253             return endrec
255 254
256 255         if diskno != 0 or disks > 1:
257 256             raise BadZipFile("zipfiles that span multiple disks are not
supported")
258 257
259 - # Assume no 'zip64 extensible data'
260 - fpin.seek(offset - sizeEndCentDir64Locator - sizeEndCentDir64, 2)
258 + offset -= sizeEndCentDir64
259 + if reloff > offset:
260 +     raise BadZipFile("Corrupt zip64 end of central directory locator")
261 + # First, check the assumption that there is no prepended data.
262 + fpin.seek(reloff)
263 + extrasz = offset - reloff
261 264 data = fpin.read(sizeEndCentDir64)
262 265 if len(data) != sizeEndCentDir64:
263 - return endrec
266 + raise OSError("Unknown I/O error")
267 + if not data.startswith(stringEndArchive64) and reloff != offset:
268 +     # Since we already have seen the Zip64 EOCD Locator, it's
269 +     # possible we got here because there is prepended data.
270 +     # Assume no 'zip64 extensible data'
271 +     fpin.seek(offset)
272 +     extrasz = 0
273 +     data = fpin.read(sizeEndCentDir64)
274 +     if len(data) != sizeEndCentDir64:
275 +         raise OSError("Unknown I/O error")
276 +     if not data.startswith(stringEndArchive64):
277 +         raise BadZipFile("Zip64 end of central directory record not found")
278 +
264 279 sig, sz, create_version, read_version, disk_num, disk_dir, \
265 280     dircount, dircount2, dirsized, diroffset = \
266 281     struct.unpack(structEndArchive64, data)
267 - if sig != stringEndArchive64:

```

268	-	<code>return endrec</code>
282	+	<code>if (diroffset + dirsize != reloff or</code>
283	+	<code>sz + 12 != sizeEndCentDir64 + extrasz):</code>
284	+	<code>raise BadZipFile("Corrupt zip64 end of central directory record")</code>
269	285	
270	286	<code># Update the original endrec using data from the ZIP64 record</code>
271	287	<code>endrec[_ECD_SIGNATURE] = sig</code>
↕		<code>@@ -275,6 +291,7 @@ def _EndRecData64(fpin, offset, endrec):</code>
275	291	<code>endrec[_ECD_ENTRIES_TOTAL] = dircount2</code>
276	292	<code>endrec[_ECD_SIZE] = dirsize</code>
277	293	<code>endrec[_ECD_OFFSET] = diroffset</code>
294	+	<code>endrec[_ECD_LOCATION] = offset - extrasz</code>
278	295	<code>return endrec</code>
279	296	
280	297	
↓		<code>@@ -308,7 +325,7 @@ def _EndRecData(fpin):</code>
↑		
308	325	<code>endrec.append(filesize - sizeEndCentDir)</code>
309	326	
310	327	<code># Try to read the "Zip64 end of central directory" structure</code>
311	-	<code>return _EndRecData64(fpin, -sizeEndCentDir, endrec)</code>
328	+	<code>return _EndRecData64(fpin, filesize - sizeEndCentDir, endrec)</code>
312	329	
313	330	<code># Either this is not a ZIP file, or it is a ZIP file with an archive</code>
314	331	<code># comment. Search the end of the file for the "end of central directory"</code>
↕		<code>@@ -332,8 +349,7 @@ def _EndRecData(fpin):</code>
332	349	<code>endrec.append(maxCommentStart + start)</code>
333	350	
334	351	<code># Try to read the "Zip64 end of central directory" structure</code>
335	-	<code>return _EndRecData64(fpin, maxCommentStart + start - filesize,</code>
336	-	<code>endrec)</code>
352	+	<code>return _EndRecData64(fpin, maxCommentStart + start, endrec)</code>
337	353	
338	354	<code># Unable to find a valid end of central directory structure</code>
339	355	<code>return None</code>
↓		<code>@@ -1427,9 +1443,6 @@ def _RealGetContents(self):</code>
↑		
1427	1443	
1428	1444	<code># "concat" is zero, unless zip was concatenated to another file</code>
1429	1445	<code>concat = endrec[_ECD_LOCATION] - size_cd - offset_cd</code>

```

1430 -         if endrec[_ECD_SIGNATURE] == stringEndArchive64:
1431 -             # If Zip64 extension structures are present, account for them
1432 -             concat -= (sizeEndCentDir64 + sizeEndCentDir64Locator)
1433 1446
1434 1447         if self.debug > 2:
1435 1448             inferred = concat + offset_cd
1436 1449
1437 1450     @@ -2047,7 +2060,7 @@ def _write_end_record(self):
1438 1451         """
1439 1452         """ would require ZIP64 extensions")
1440 1453         zip64endrec = struct.pack(
1441 1454             structEndArchive64, stringEndArchive64,
1442 1455             44, 45, 45, 0, 0, centDirCount, centDirCount,
1443 1456             sizeEndCentDir64 - 12, 45, 45, 0, 0, centDirCount,
1444 1457             centDirCount,
1445 1458             centDirSize, centDirOffset)
1446 1459         self.fp.write(zip64endrec)
1447 1460
1448 1461
1449 1462
1450 1463
1451 1464
1452 1465
1453 1466
1454 1467
1455 1468
1456 1469
1457 1470
1458 1471
1459 1472
1460 1473
1461 1474
1462 1475
1463 1476
1464 1477
1465 1478
1466 1479
1467 1480
1468 1481
1469 1482
1470 1483
1471 1484
1472 1485
1473 1486
1474 1487
1475 1488
1476 1489
1477 1490
1478 1491
1479 1492
1480 1493
1481 1494
1482 1495
1483 1496
1484 1497
1485 1498
1486 1499
1487 1500
1488 1501
1489 1502
1490 1503
1491 1504
1492 1505
1493 1506
1494 1507
1495 1508
1496 1509
1497 1510
1498 1511
1499 1512
1500 1513
1501 1514
1502 1515
1503 1516
1504 1517
1505 1518
1506 1519
1507 1520
1508 1521
1509 1522
1510 1523
1511 1524
1512 1525
1513 1526
1514 1527
1515 1528
1516 1529
1517 1530
1518 1531
1519 1532
1520 1533
1521 1534
1522 1535
1523 1536
1524 1537
1525 1538
1526 1539
1527 1540
1528 1541
1529 1542
1530 1543
1531 1544
1532 1545
1533 1546
1534 1547
1535 1548
1536 1549
1537 1550
1538 1551
1539 1552
1540 1553
1541 1554
1542 1555
1543 1556
1544 1557
1545 1558
1546 1559
1547 1560
1548 1561
1549 1562
1550 1563
1551 1564
1552 1565
1553 1566
1554 1567
1555 1568
1556 1569
1557 1570
1558 1571
1559 1572
1560 1573
1561 1574
1562 1575
1563 1576
1564 1577
1565 1578
1566 1579
1567 1580
1568 1581
1569 1582
1570 1583
1571 1584
1572 1585
1573 1586
1574 1587
1575 1588
1576 1589
1577 1590
1578 1591
1579 1592
1580 1593
1581 1594
1582 1595
1583 1596
1584 1597
1585 1598
1586 1599
1587 1600
1588 1601
1589 1602
1590 1603
1591 1604
1592 1605
1593 1606
1594 1607
1595 1608
1596 1609
1597 1610
1598 1611
1599 1612
1600 1613
1601 1614
1602 1615
1603 1616
1604 1617
1605 1618
1606 1619
1607 1620
1608 1621
1609 1622
1610 1623
1611 1624
1612 1625
1613 1626
1614 1627
1615 1628
1616 1629
1617 1630
1618 1631
1619 1632
1620 1633
1621 1634
1622 1635
1623 1636
1624 1637
1625 1638
1626 1639
1627 1640
1628 1641
1629 1642
1630 1643
1631 1644
1632 1645
1633 1646
1634 1647
1635 1648
1636 1649
1637 1650
1638 1651
1639 1652
1640 1653
1641 1654
1642 1655
1643 1656
1644 1657
1645 1658
1646 1659
1647 1660
1648 1661
1649 1662
1650 1663
1651 1664
1652 1665
1653 1666
1654 1667
1655 1668
1656 1669
1657 1670
1658 1671
1659 1672
1660 1673
1661 1674
1662 1675
1663 1676
1664 1677
1665 1678
1666 1679
1667 1680
1668 1681
1669 1682
1670 1683
1671 1684
1672 1685
1673 1686
1674 1687
1675 1688
1676 1689
1677 1690
1678 1691
1679 1692
1680 1693
1681 1694
1682 1695
1683 1696
1684 1697
1685 1698
1686 1699
1687 1700
1688 1701
1689 1702
1690 1703
1691 1704
1692 1705
1693 1706
1694 1707
1695 1708
1696 1709
1697 1710
1698 1711
1699 1712
1700 1713
1701 1714
1702 1715
1703 1716
1704 1717
1705 1718
1706 1719
1707 1720
1708 1721
1709 1722
1710 1723
1711 1724
1712 1725
1713 1726
1714 1727
1715 1728
1716 1729
1717 1730
1718 1731
1719 1732
1720 1733
1721 1734
1722 1735
1723 1736
1724 1737
1725 1738
1726 1739
1727 1740
1728 1741
1729 1742
1730 1743
1731 1744
1732 1745
1733 1746
1734 1747
1735 1748
1736 1749
1737 1750
1738 1751
1739 1752
1740 1753
1741 1754
1742 1755
1743 1756
1744 1757
1745 1758
1746 1759
1747 1760
1748 1761
1749 1762
1750 1763
1751 1764
1752 1765
1753 1766
1754 1767
1755 1768
1756 1769
1757 1770
1758 1771
1759 1772
1760 1773
1761 1774
1762 1775
1763 1776
1764 1777
1765 1778
1766 1779
1767 1780
1768 1781
1769 1782
1770 1783
1771 1784
1772 1785
1773 1786
1774 1787
1775 1788
1776 1789
1777 1790
1778 1791
1779 1792
1780 1793
1781 1794
1782 1795
1783 1796
1784 1797
1785 1798
1786 1799
1787 1800
1788 1801
1789 1802
1790 1803
1791 1804
1792 1805
1793 1806
1794 1807
1795 1808
1796 1809
1797 1810
1798 1811
1799 1812
1800 1813
1801 1814
1802 1815
1803 1816
1804 1817
1805 1818
1806 1819
1807 1820
1808 1821
1809 1822
1810 1823
1811 1824
1812 1825
1813 1826
1814 1827
1815 1828
1816 1829
1817 1830
1818 1831
1819 1832
1820 1833
1821 1834
1822 1835
1823 1836
1824 1837
1825 1838
1826 1839
1827 1840
1828 1841
1829 1842
1830 1843
1831 1844
1832 1845
1833 1846
1834 1847
1835 1848
1836 1849
1837 1850
1838 1851
1839 1852
1840 1853
1841 1854
1842 1855
1843 1856
1844 1857
1845 1858
1846 1859
1847 1860
1848 1861
1849 1862
1850 1863
1851 1864
1852 1865
1853 1866
1854 1867
1855 1868
1856 1869
1857 1870
1858 1871
1859 1872
1860 1873
1861 1874
1862 1875
1863 1876
1864 1877
1865 1878
1866 1879
1867 1880
1868 1881
1869 1882
1870 1883
1871 1884
1872 1885
1873 1886
1874 1887
1875 1888
1876 1889
1877 1890
1878 1891
1879 1892
1880 1893
1881 1894
1882 1895
1883 1896
1884 1897
1885 1898
1886 1899
1887 1900
1888 1901
1889 1902
1890 1903
1891 1904
1892 1905
1893 1906
1894 1907
1895 1908
1896 1909
1897 1910
1898 1911
1899 1912
1900 1913
1901 1914
1902 1915
1903 1916
1904 1917
1905 1918
1906 1919
1907 1920
1908 1921
1909 1922
1910 1923
1911 1924
1912 1925
1913 1926
1914 1927
1915 1928
1916 1929
1917 1930
1918 1931
1919 1932
1920 1933
1921 1934
1922 1935
1923 1936
1924 1937
1925 1938
1926 1939
1927 1940
1928 1941
1929 1942
1930 1943
1931 1944
1932 1945
1933 1946
1934 1947
1935 1948
1936 1949
1937 1950
1938 1951
1939 1952
1940 1953
1941 1954
1942 1955
1943 1956
1944 1957
1945 1958
1946 1959
1947 1960
1948 1961
1949 1962
1950 1963
1951 1964
1952 1965
1953 1966
1954 1967
1955 1968
1956 1969
1957 1970
1958 1971
1959 1972
1960 1973
1961 1974
1962 1975
1963 1976
1964 1977
1965 1978
1966 1979
1967 1980
1968 1981
1969 1982
1970 1983
1971 1984
1972 1985
1973 1986
1974 1987
1975 1988
1976 1989
1977 1990
1978 1991
1979 1992
1980 1993
1981 1994
1982 1995
1983 1996
1984 1997
1985 1998
1986 1999
1987 2000
1988 2001
1989 2002
1990 2003
1991 2004
1992 2005
1993 2006
1994 2007
1995 2008
1996 2009
1997 2010
1998 2011
1999 2012
2000 2013
2001 2014
2002 2015
2003 2016
2004 2017
2005 2018
2006 2019
2007 2020
2008 2021
2009 2022
2010 2023
2011 2024
2012 2025
2013 2026
2014 2027
2015 2028
2016 2029
2017 2030
2018 2031
2019 2032
2020 2033
2021 2034
2022 2035
2023 2036
2024 2037
2025 2038
2026 2039
2027 2040
2028 2041
2029 2042
2030 2043
2031 2044
2032 2045
2033 2046
2034 2047
2035 2048
2036 2049
2037 2050
2038 2051
2039 2052
2040 2053
2041 2054
2042 2055
2043 2056
2044 2057
2045 2058
2046 2059
2047 2060
2048 2061
2049 2062
2050 2063
2051 2064
2052 2065
2053 2066
2054 2067
2055 2068
2056 2069
2057 2070
2058 2071
2059 2072
2060 2073
2061 2074
2062 2075
2063 2076
2064 2077
2065 2078
2066 2079
2067 2080
2068 2081
2069 2082
2070 2083
2071 2084
2072 2085
2073 2086
2074 2087
2075 2088
2076 2089
2077 2090
2078 2091
2079 2092
2080 2093
2081 2094
2082 2095
2083 2096
2084 2097
2085 2098
2086 2099
2087 2100
2088 2101
2089 2102
2090 2103
2091 2104
2092 2105
2093 2106
2094 2107
2095 2108
2096 2109
2097 2110
2098 2111
2099 2112
2100 2113
2101 2114
2102 2115
2103 2116
2104 2117
2105 2118
2106 2119
2107 2120
2108 2121
2109 2122
2110 2123
2111 2124
2112 2125
2113 2126
2114 2127
2115 2128
2116 2129
2117 2130
2118 2131
2119 2132
2120 2133
2121 2134
2122 2135
2123 2136
2124 2137
2125 2138
2126 2139
2127 2140
2128 2141
2129 2142
2130 2143
2131 2144
2132 2145
2133 2146
2134 2147
2135 2148
2136 2149
2137 2150
2138 2151
2139 2152
2140 2153
2141 2154
2142 2155
2143 2156
2144 2157
2145 2158
2146 2159
2147 2160
2148 2161
2149 2162
2150 2163
2151 2164
2152 2165
2153 2166
2154 2167
2155 2168
2156 2169
2157 2170
2158 2171
2159 2172
2160 2173
2161 2174
2162 2175
2163 2176
2164 2177
2165 2178
2166 2179
2167 2180
2168 2181
2169 2182
2170 2183
2171 2184
2172 2185
2173 2186
2174 2187
2175 2188
2176 2189
2177 2190
2178 2191
2179 2192
2180 2193
2181 2194
2182 2195
2183 2196
2184 2197
2185 2198
2186 2199
2187 2200
2188 2201
2189 2202
2190 2203
2191 2204
2192 2205
2193 2206
2194 2207
2195 2208
2196 2209
2197 2210
2198 2211
2199 2212
2200 2213
2201 2214
2202 2215
2203 2216
2204 2217
2205 2218
2206 2219
2207 2220
2208 2221
2209 2222
2210 2223
2211 2224
2212 2225
2213 2226
2214 2227
2215 2228
2216 2229
2217 2230
2218 2231
2219 2232
2220 2233
2221 2234
2222 2235
2223 2236
2224 2237
2225 2238
2226 2239
2227 2240
2228 2241
2229 2242
2230 2243
2231 2244
2232 2245
2233 2246
2234 2247
2235 2248
2236 2249
2237 2250
2238 2251
2239 2252
2240 2253
2241 2254
2242 2255
2243 2256
2244 2257
2245 2258
2246 2259
2247 2260
2248 2261
2249 2262
2250 2263
2251 2264
2252 2265
2253 2266
2254 2267
2255 2268
2256 2269
2257 2270
2258 2271
2259 2272
2260 2273
2261 2274
2262 2275
2263 2276
2264 2277
2265 2278
2266 2279
2267 2280
2268 2281
2269 2282
2270 2283
2271 2284
2272 2285
2273 2286
2274 2287
2275 2288
2276 2289
2277 2290
2278 2291
2279 2292
2280 2293
2281 2294
2282 2295
2283 2296
2284 2297
2285 2298
2286 2299
2287 2300
2288 2301
2289 2302
2290 2303
2291 2304
2292 2305
2293 2306
2294 2307
2295 2308
2296 2309
2297 2310
2298 2311
2299 2312
2300 2313
2301 2314
2302 2315
2303 2316
2304 2317
2305 2318
2306 2319
2307 2320
2308 2321
2309 2322
2310 2323
2311 2324
2312 2325
2313 2326
2314 2327
2315 2328
2316 2329
2317 2330
2318 2331
2319 2332
2320 2333
2321 2334
2322 2335
2323 2336
2324 2337
2325 2338
2326 2339
2327 2340
2328 2341
2329 2342
2330 2343
2331 2344
2332 2345
2333 2346
2334 2347
2335 2348
2336 2349
2337 2350
2338 2351
2339 2352
2340 2353
2341 2354
2342 2355
2343 2356
2344 2357
2345 2358
2346 2359
2347 2360
2348 2361
2349 2362
2350 2363
2351 2364
2352 2365
2353 2366
2354 2367
2355 2368
2356 2369
2357 2370
2358 2371
2359 2372
2360 2373
2361 2374
2362 2375
2363 2376
2364 2377
2365 2378
2366 2379
2367 2380
2368 2381
2369 2382
2370 2383
2371 2384
2372 2385
2373 2386
2374 2387
2375 2388
2376 2389
2377 2390
2378 2391
2379 2392
2380 2393
2381 2394
2382 2395
2383 2396
2384 2397
2385 2398
2386 2399
2387 2400
2388 2401
2389 2402
2390 2403
2391 2404
2392 2405
2393 2406
2394 2407
2395 2408
2396 2409
2397 2410
2398 2411
2399 2412
2400 2413
2401 2414
2402 2415
2403 2416
2404 2417
2405 2418
2406 2419
2407 2420
2408 2421
2409 2422
2410 2423
2411 2424
2412 2425
2413 2426
2414 2427
2415 2428
2416 2429
2417 2430
2418 2431
2419 2432
2420 2433
2421 2434
2422 2435
2423 2436
2424 2437
2425 2438
2426 2439
2427 2440
2428 2441
2429 2442
2430 2443
2431 2444
2432 2445
2433 2446
2434 2447
2435 2448
2436 2449
2437 2450
2438 2451
2439 2452
2440 2453
2441 2454
2442 2455
2443 2456
2444 2457
2445 2458
2446 2459
2447 2460
2448 2461
2449 2462
2450 2463
2451 2464
2452 2465
2453 2466
2454 2467
2455 2468
2456 2469
2457 2470
2458 2471
2459 2472
2460 2473
2461 2474
2462 2475
2463 2476
2464 2477
2465 2478
2466 2479
2467 2480
2468 2481
2469 2482
2470 2483
2471 2484
2472 2485
2473 2486
2474 2487
2475 2488
2476 2489
2477 2490
2478 2491
2479 2492
2480 2493
2481 2494
2482 2495
2483 2496
2484 2497
2485 2498
2486 2499
2487 2500
2488 2501
2489 2502
2490 2503
2491 2504
2492 2505
2493 2506
2494 2507
2495 2508
2496 2509
2497 2510
2498 2511
2499 2512
2500 2513
2501 2514
2502 2515
2503 2516
2504 2517
2505 2518
2506 2519
2507 2520
2508 2521
2509 2522
2510 2523
2511 2524
2512 2525
2513 2526
2514 2527
2515 2528
2516 2529
2517 2530
2518 2531
2519 2532
2520 2533
2521 2534
2522 2535
2523 2536
2524 2537
2525 2538
2526 2539
2527 2540
2528 2541
2529 2542
2530 2543
2531 2544
2532 2545
2533 2546
2534 2547
2535 2548
2536 2549
2537 2550
2538 2551
2539 2552
2540 2553
2541 2554
2542 2555
2543 2556
2544 2557
2545 2558
2546 2559
2547 2560
2548 2561
2549 2562
2550 2563
2551 2564
2552 2565
2553 2566
2554 2567
2555 2568
2556 2569
2557 2570
2558 2571
2559 2572
2560 2573
2561 2574
2562 2575
2563 2576
2564 2577
2565 2578
2566 2579
2567 2580
2568 2581
2569 2582
2570 2583
2571 2584
2572 2585
2573 2586
2574 2587
2575 2588
2576 2589
2577 2590
2578 2591
2579 2592
2580 2593
2581 2594
2582 2595
2583 2596
2584 2597
2585 2598
2586 2599
2587 2600
2588 2601
2589 2602
2590 2603
2591 2604
2592 2605
2593 2606
2594 2607
2595 2608
2596 2609
2597 2610
2598 2611
2599 2612
2600 2613
2601 2614
2602 2615
2603 2616
2604 2617
2605 2618
2606 2619
2607 2620
2608 2621
2609 2622
2610 2623
2611 2624
2612 2625
2613 2626
2614 2627
2615 2628
2616 2629
2617 2630
2618 2631
2619 2632
2620 2633
2621 2634
2622 2635
2623 2636
2624 2637
2625 2638
2626 2639
2627 2640
2628 2641
2629 2642
2630 2643
2631 2644
2632 2645
2633 2646
2634
```